Task 2: Scaling the WordFreq Application

1. Wordfreq Application Overview with AWS

The Wordfreq Application is a piece of custom software written in the Go computer language that can read and change text data. The main job of this programme is to find out how many times words show up in various text files. These words are used a lot in these files, so this tells you something important. People know this app for how quickly and well it works, which are important parts of Go's performance-driven design. This app lets users add different text files, and then it quickly scans them by using Go's powerful parallel features to handle very big datasets. The end result is a full list of words ranked by how often they appear.

To start with, set up an Amazon Web Services (AWS)-based word frequency analysis application in the cloud. AWS services like Elastic Cloud Computing (EC2), Simple Storage Service (S3), Simple Queue Service (SQS), and Simple Notification Service (SNS) are used in the setup. The process begins with setting up an EC2 server with specific settings, like using Ubuntu Server 22.04 LTS and creating a security group and an Identity Access Management (IAM) role. Two S3 buckets are set up so that text files can be uploaded and processed, and two SQS queues are set up to handle job requests and results. It is possible to set up a Subject Notification Service (SNS) topic just for sending alerts about how files are being processed. After that, the EC2 server is set up with necessary software like AWS CLI, the Go programming language, and the application code that was pulled from an S3 bucket. Users can make changes to script files so that they work with their AWS setup without any problems.

To test how well an application works, a series of steps are taken, including moving text files to S3 buckets, keeping an eye on the worker application's work, and checking the results in DynamoDB and through email alerts. In the end setup, the worker application is set up as a Linux service to make sure it works without interruption. One needs to keep a close eye on how the EC2 servers are being used so that they can

effectively control costs. Also, making sure to shut down instances when they're not being used.

2. Design and Implementing Autoscaling

a) Launching an EC2 server: Launching an EC2 server is the first step in setting up AWS. First, give your server a name and choose the Ubuntu Server AMI as an operating system that will work well. It is best to use the t2.micro instance type for general apps. For safe entry through PuTTY, make a key pair. Use the wordfreq security group to set up your protection. In advanced settings, find the IAM instance profile and click on EMR_EC2_DefaultRole. With these settings, you can start up your EC2 server.

b) Making S3 Buckets: Make two S3 buckets before going to data storage. One can upload and store text files from your computer in the "Uploading S3 Bucket." 'Processing S3 Bucket' holds data that is ready to be processed. To send reports to the wordfreq SQS queue and the email, turn on upload notifications in the Uploading bucket. Make these buckets in the S3 service and give each one a unique name.

c) Setting up SQS Queues: Next, create a new SQS queue for jobs that will handle files. To get a lot of transactions done, choose the "Standard" queue type. Make the names of the queues clear so they are easy to handle. You can set access policies to standard for access and change the JSON configuration to meet individual needs. Write down the URLs of the job and results queues so that they can be used in the future.

d) Adding Data to the Uploading Bucket: Adding data to the "Uploading S3 Bucket," particularly 120 text files. This step is very important because it gives your system the first set of data to work

with. Make sure that all of the files are uploaded properly so that the process goes smoothly.

e) Creating a Launch Template: Finally, create a launch template. Give your template a name and choose the AMI that was used before to make sure that the setting stays the same. Pick the right instance type and add your security group and the learnerlab keypair. Choose EMR_EC2_DefaultRole to get the rights you need and turn on monitoring in CloudWatch.

f) Setting up alarms in CloudWatch Monitoring: Set up alarms in CloudWatch to keep an eye on your autoscaling more effectively. These alarms tell the autoscaling group whether to launch or terminate instances based on certain conditions. To set this up, use the queue metrics, taking particular consideration to the "approximate number of messages visible" number from the job queue. This number is the best way to understand how busy the queue is. Set up two alarms. One will trigger when there are more than 25 messages that can be seen, which will force a new instance to be added. The second one would fire when the count drops below 75, which suggests that instances may need to be lowered.

g) Making an Autoscaling Group: The next step is to make an Autoscaling Group (ASG). This group is very important because it controls when instances start up and end, based on your launch template, which is linked to a specific instance type. Start by picking out the right launch design. Next, you can either give approval for a Virtual Private Cloud (VPC) that has already been set up or create a new one. Pick the availability zones where your instances will run. It is very important to set the preferred, minimum, and maximum number of instances in the group. Adding tags is optional, but it can help organise things into groups. Finally, set up the Auto Scaling Group (ASG).

(Https://Aws.Amazon.Com/Ec2/Autoscaling/Getting-Started/, n.d.)

h) Starting to put the Dynamic Scaling Policy into action: When it comes to growing instances, you can do a number of different things. Although simple scaling reacts to alerts, step scaling gives you more control by letting you set exact times for instances to start and stop. For this task, one should use step scaling. Set up two dynamic scaling policies. One will lower the number of instances linked to the alert when the threshold is below 75, and the other will raise the number of instances when the threshold is higher than 25.

i) Clearing the queues and processing bucket: It is important to get rid of old data before starting autoscaling again. Get rid of all the items in the job queue by purging it. Next, empty the processing bucket, which likely has data from earlier auto-scaling tasks. This step makes sure that the new autoscaling process starts over from scratch.

j) As the last step in the setup process, you need to use PuTTY to connect to your EC2 server. Use the Public IPv4 DNS address and the key pair that you made earlier. Run the code to move data from the uploading bucket to the processing bucket once the link is made. As soon as this process starts, S3 will start sending you emails to let you know.

k) Monitoring phase: At this point, keep an eye on the whole process. Pick the job queue from the SQS queues and keep an eye on the "number of messages in flight." The goal is for this number to go down to zero, which would mean that all texts have been processed. Keep an eye on how often instances are launched and terminated in response to your dynamic scaling policies. This monitoring lets you know how well and quickly your autoscaling setup is working.

3. Using t3.medium instance and dynamic scaling policies to optimize performance:

After careful consideration, the t3.medium instance has emerged as the ideal choice for handling tasks, completing them in an efficient 7 minutes and 55 seconds. To make the autoscaling system respond best to changes in workload, exact alarm thresholds and dynamic scaling policies have been set up.

The alarm settings are crucial for maintaining balance. They are programmed to activate when the number of visible messages in the job queue exceeds 25 and again when it falls below 75. This range is selected to ensure the system scales appropriately based on the workload.

The dynamic scaling policies are designed to match the fluctuating demand precisely: For scaling up, the system initially launches 1 instance when the message count is between 25-50. As the workload increases, an additional instance is added for a message range of 50-75. For a more significant increase, specifically when messages range between 75-100, the system launches 2 more instances. If the messages exceed 100, it deploys 2 additional instances to handle the load efficiently.

Conversely, for scaling down, the system terminates 2 instances when the message count drops to the 100-75 range. If the messages continue to decrease, falling between 75-50, it further reduces 2 instances. For a message count between 50-25, the system scales down by 1 instance. Finally, when the messages drop below 25, it terminates one more instance to optimize resource usage.

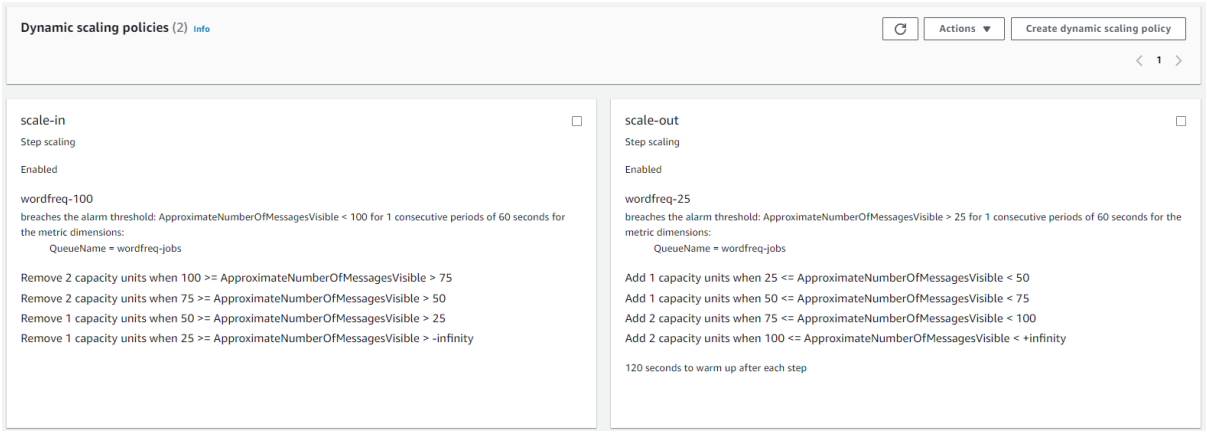

Figure 1: Configuring Dynamic Scaling policies in an Autoscaling group

4. Overview of tests and results

- The t3 general-purpose instance series is great for word frequency applications with basic needs because it has a good mix of CPU, memory, and networking resources. These situations work well and don't cost as much as more specialised options. The burstable speed of t3 instances is perfect for the periodic processing surges that come up with text data analysis. Their scalability and adaptability also make it easy to handle a lot of work. Overall, the t3 series can be used for a lot of different things, is cheap, and has a lot of resources for word frequency application computing and data processing. Therefore, a comparison of 3 same category instances has been tested for the task. The following pages have the results for the instance type: t3.nano

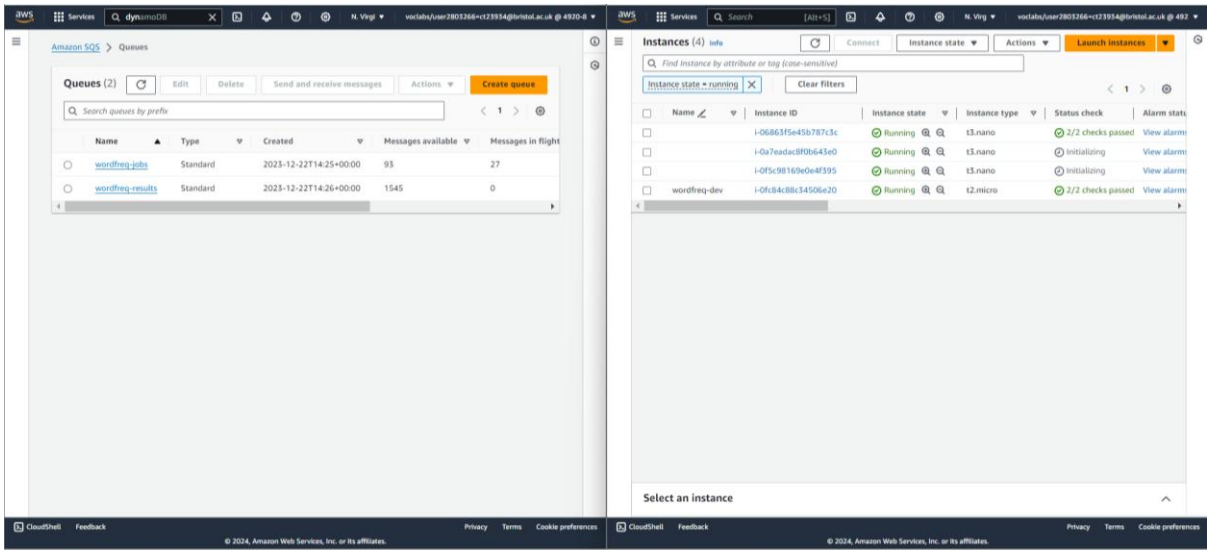

Figure 2: SQS showing message status, EC2 instance page showing launched and terminated

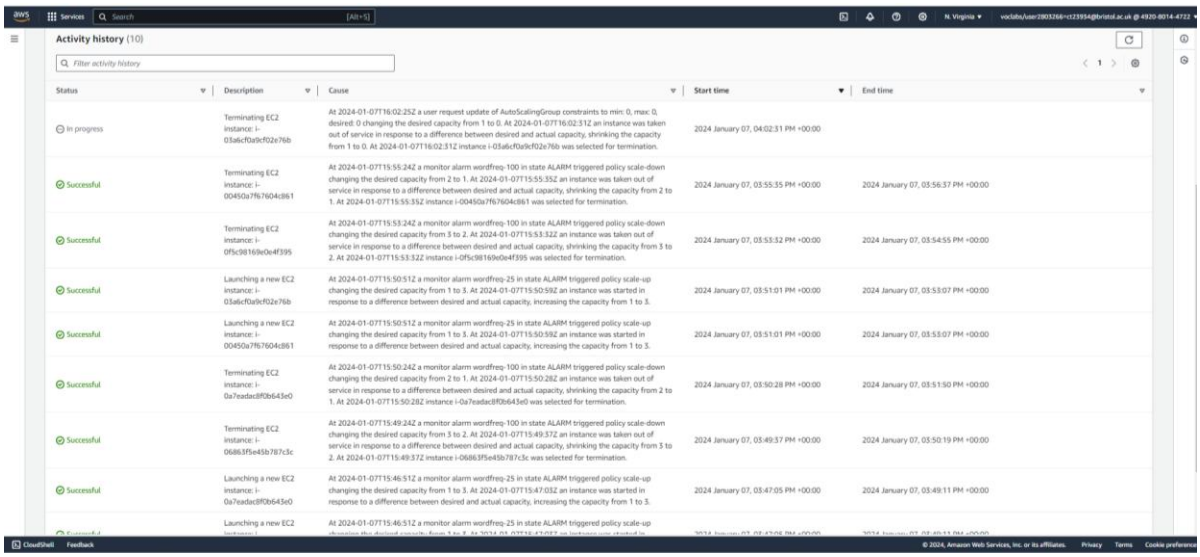

Figure 3: Autoscaling Group log showing Launching and terminating of Instances

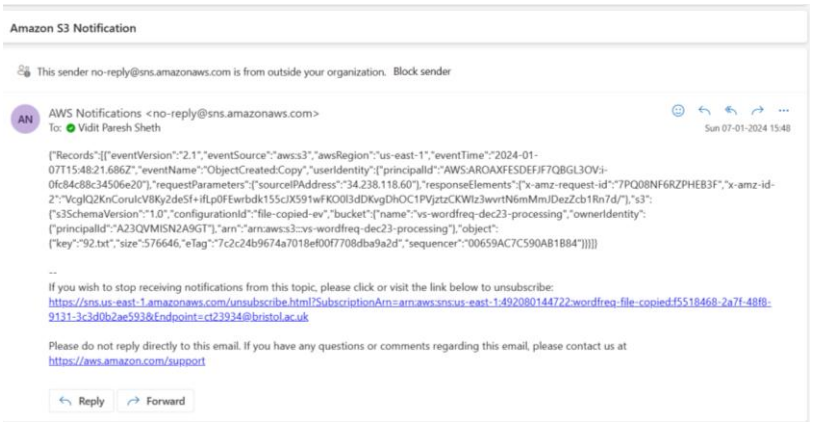

Figure 4: email received from Amazon S3 Notification

Instance type: t3.medium

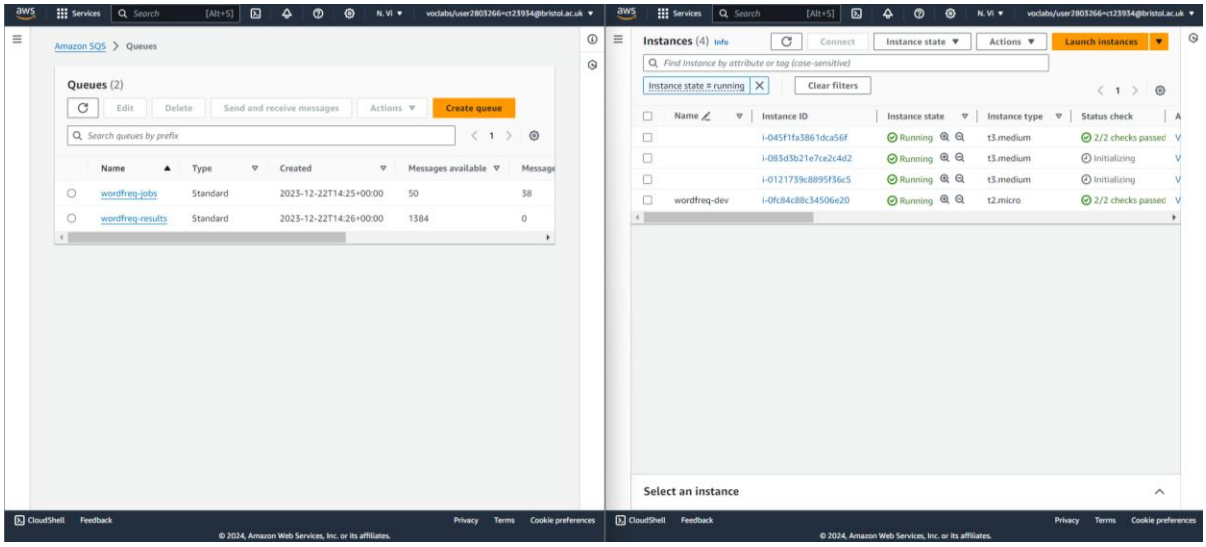

Figure 5: SQS showing message status, EC2 instance page showing launched and terminated

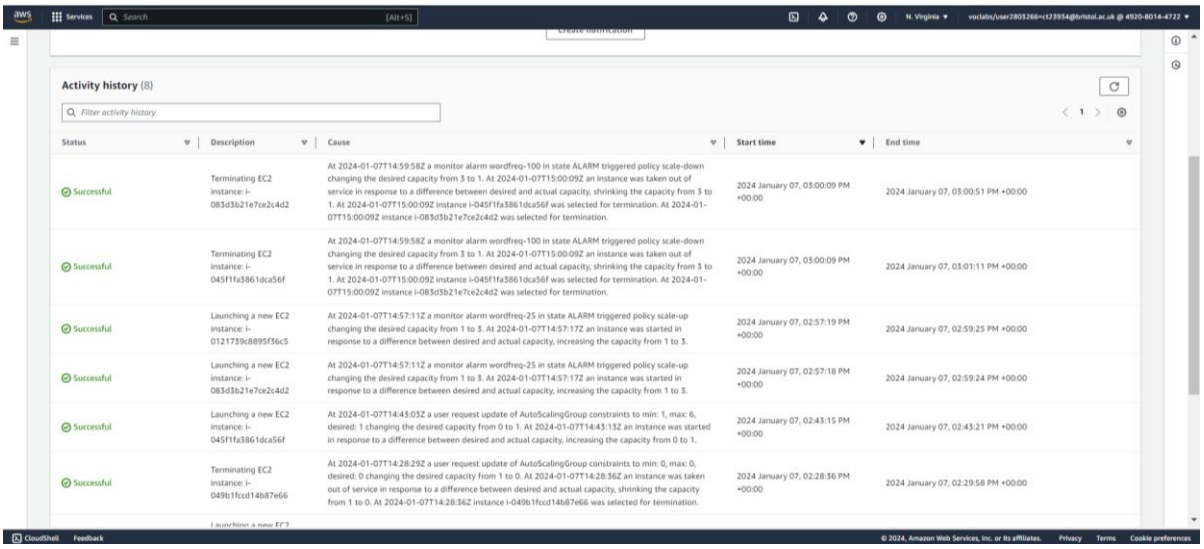

Figure 6: Autoscaling Group log showing Launching and terminating of Instances

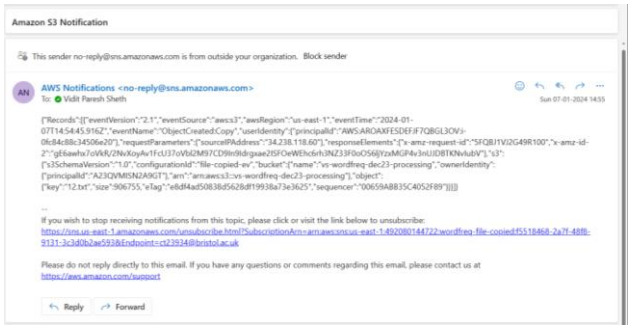

Figure 7: email received from Amazon S3 Notification

Instance type: t3a.large

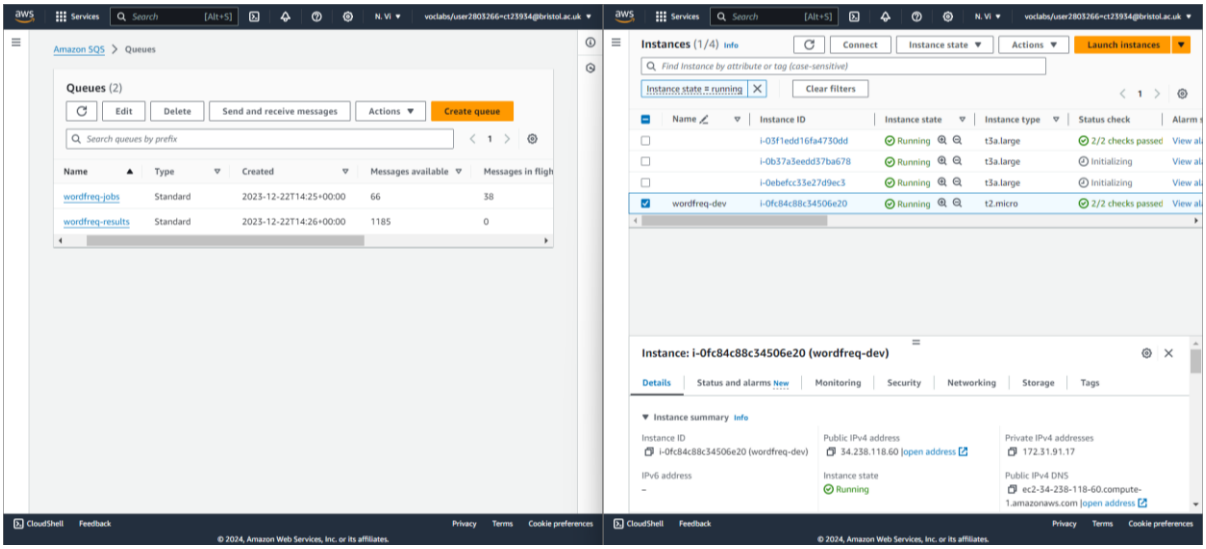

Figure 8: SQS showing message status, EC2 instance page showing launched and terminated

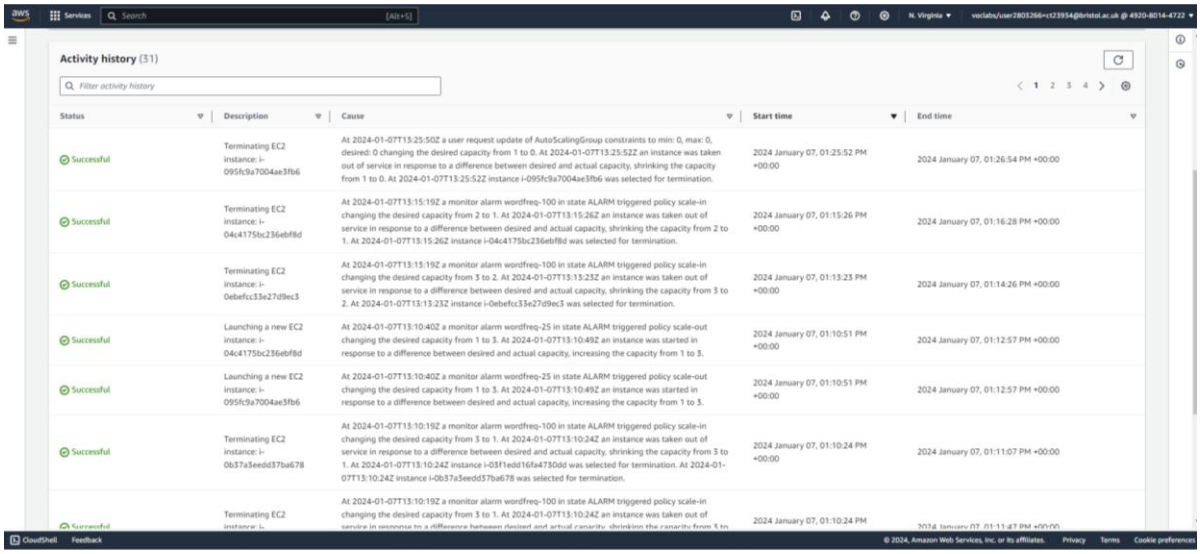

Figure 9: Autoscaling Group log showing Launching and terminating of Instances

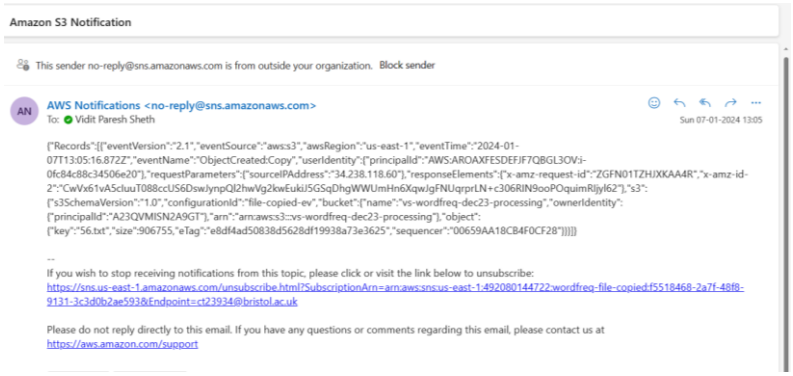

Figure 10: email received from Amazon S3 Notification

- The compute-optimized nature of AWS C5 instance types make them good for processing large amounts of text data for word frequency apps. These instances are a good mix of price and speed, and they come in different sizes to handle different amounts of work. Their network performance makes it faster for data to be sent, which helps applications that receive data from far away. C5 servers can be customised and work with AWS tools and services. Therefore, testing for instance type c5.large are as follows:

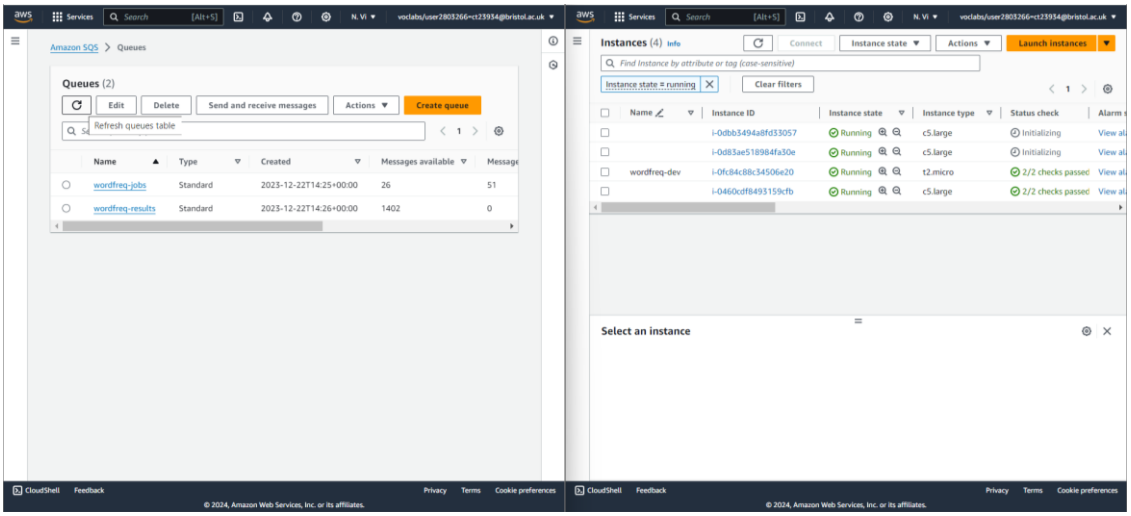

Figure 11: SQS showing message status, EC2 instance page showing launched and terminated

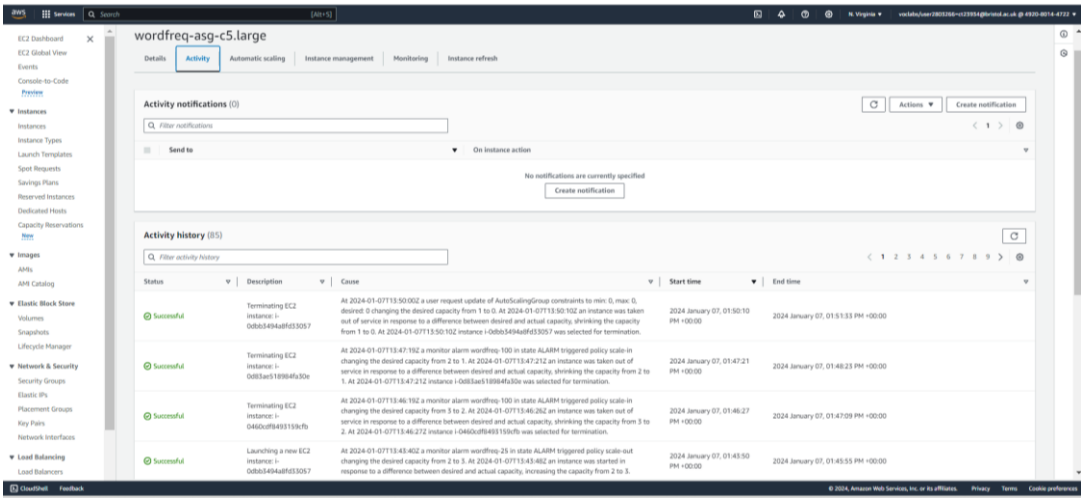

Figure 12: Autoscaling Group log showing Launching and terminating of Instances

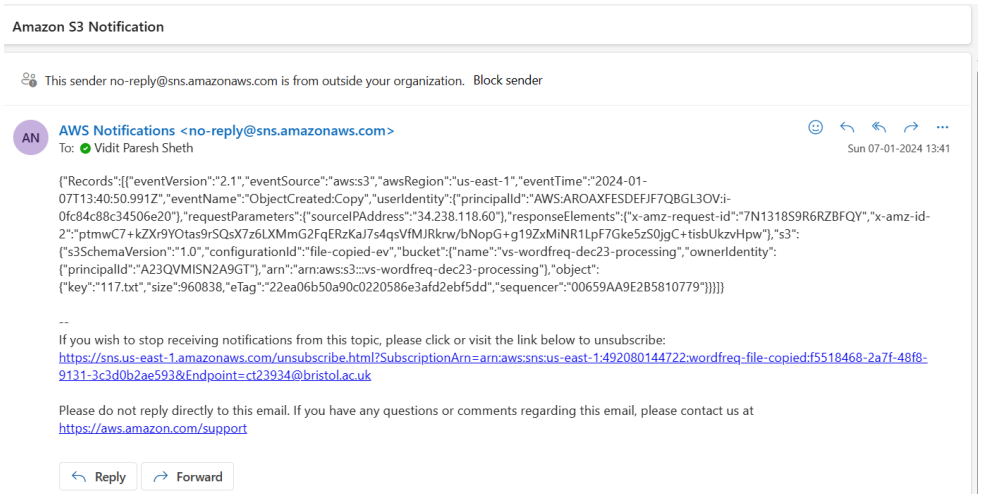

Figure 13: email received from Amazon S3 Notification

- Generally useful M5 instances in AWS have a good mix of CPU, memory, and network features, which makes them a good choice for word frequency apps. M5 instances are cheap and powerful enough to process text and figure out word frequencies on medium-sized to big datasets. There are different sizes to fit your needs. When it comes to getting word frequency data, M5 cases work well with networks. Because they can connect to AWS services and tools, it's easy to use them in an AWS context. This makes them flexible for word frequency use cases. (*Https://Aws.Amazon.Com/Ec2/Instance-Types/*, n.d.) Therefore, testing for instance type m5.large are as follows:

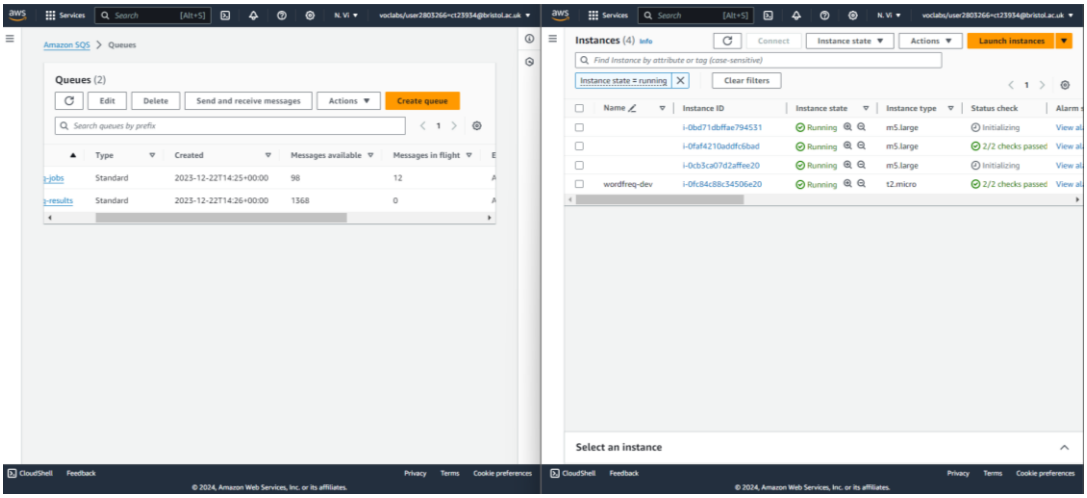

Figure 14: SQS showing message status, EC2 instance page showing launched and terminated

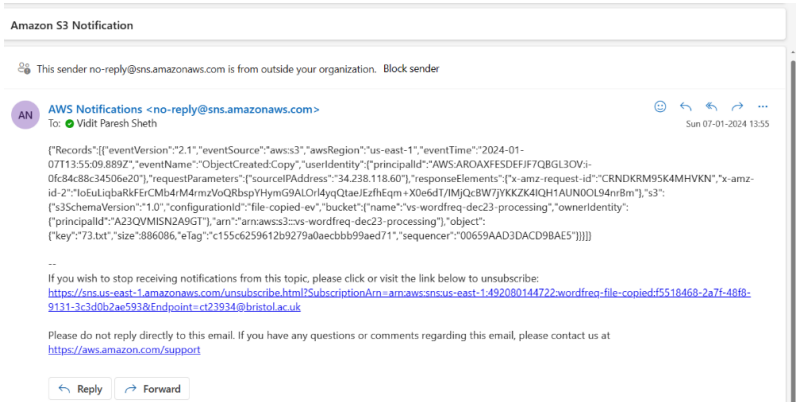

Figure 15: email received from Amazon S3 Notification

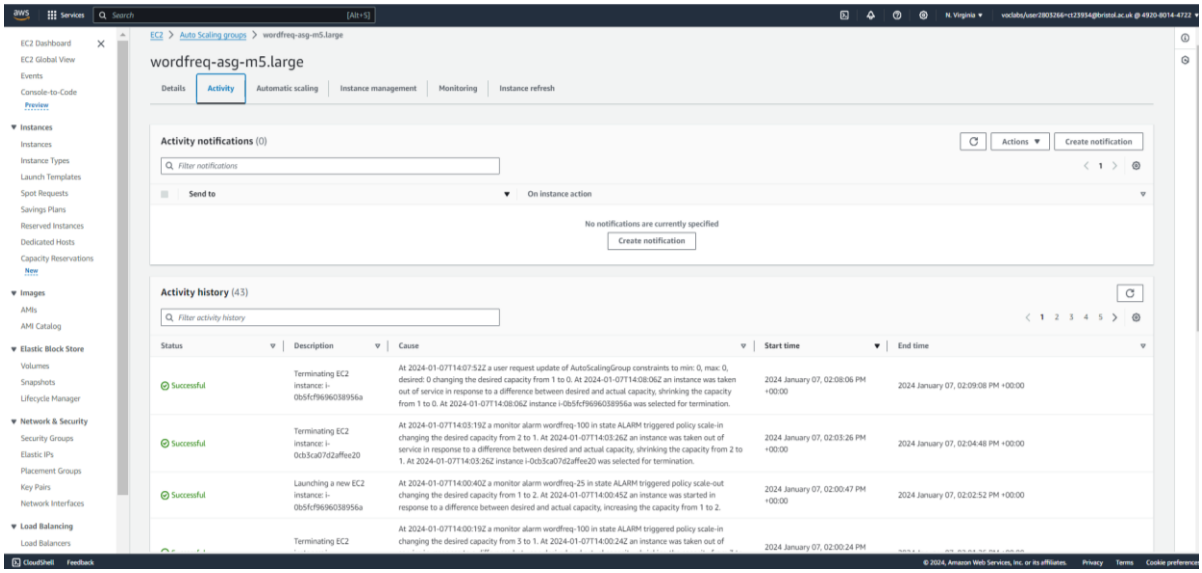

Figure 16: Autoscaling Group log showing Launching and terminating of Instances

5. Comparing different Instance types and costs for Autoscaling:

Instance Type	Price per Hour	Total Time Elapsed	Cost
General Purpose [t3.nano]	$0.0052	12min 05sec	$0.0010556
General Purpose [T3.medium]	$0.0416	7min 55sec	$0.005375
General Purpose [T3a.large]	$0.0752	10min 27sec	$0.013092
Compute Optimized [c5.large]	$0.085	7min 42sec	$0.0109158
Memory Optimized [m5.large]	$0.096	10min 42sec	$0.0171488

(*Https://Aws.Amazon.Com/Ec2/Pricing/on-Demand/*, n.d.)

The ideal choice of the most appropriate instance depends on your personal requirements and specifications. Data regarding the costs and duration associated with different types of occurrences have been compared in the table above. To arrive at a decision, it is crucial to consider many factors, including cost, time, computational power, and memory capacity, which are important for your particular task.

It's important to think about the following things:

Cost: The General Purpose [t3.nano] instance type is the most cost-effective choice if saving money is very important.

Time: General Purpose [t3.medium] type has shorter elapsed times and can help finish chores faster than t3.nano.

Balanced: The General Purpose [T3.medium] option is a good choice for a good balance of price and performance, as it has a fair price and a fairly quick working time.

6. Optimising WordFreq Architecture

The AWS architecture can be improved with various services to which the learner lab does not have access. Following can be taken into consideration to:

Increase resilience and handling component failure:

- Using AWS Lambda for programme computing. Lambda ensures the technology behind is always available and can evolve. When a part breaks, it repairs itself instantaneously. Auto-scaling Lambda functions helps them manage varied workloads.
- Cost and time optimization is another feature of lambda where the serverless architecture plays an important role.

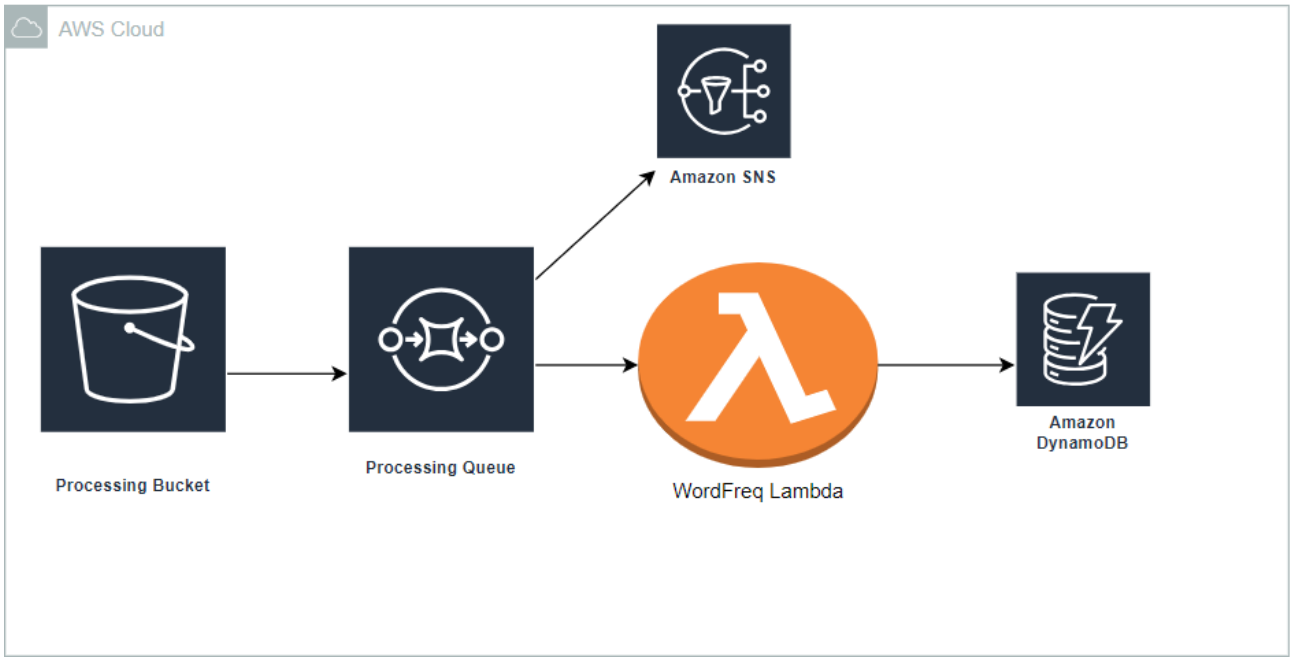

Figure 17: Architecture with Lambda

Backup crucial data long-term:

- Storing data on AWS S3 buckets as they are ideal for application logs.
- Setup S3 Lifecycle Policies: S3 lifecycle rules migrate older data to AWS Glacier for long-term storage. This reduces storage costs and protects data. The data is transferred from the standard S3 bucket to a not frequently accessed bucket after certain number of user defined days, which is further sent to AWS Glacier and finally deleted. The cost-effective service helps user define the number of days to wait before any transfers.

- Backup DynamoDB: Built-in backup and restoration tools in DynamoDB allow frequent backups of critical application data.

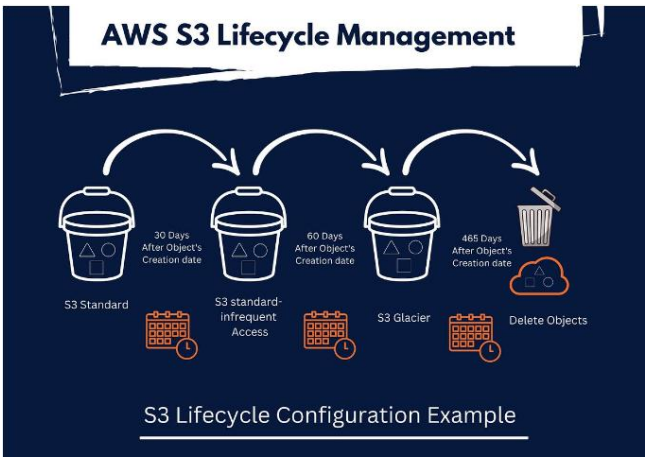

Figure 18: AWS S3 Lifecycle Management

Cost-effective and effective application for occasional use:

- AWS Lambda's serverless architecture lets you pay only for what you use. Lambda functions are cost-effective for apps with shifting duties because they're free while not in use.
- Amazon API Gateway controls and displays the app's API interfaces. Access control and use monitoring are easy with this.

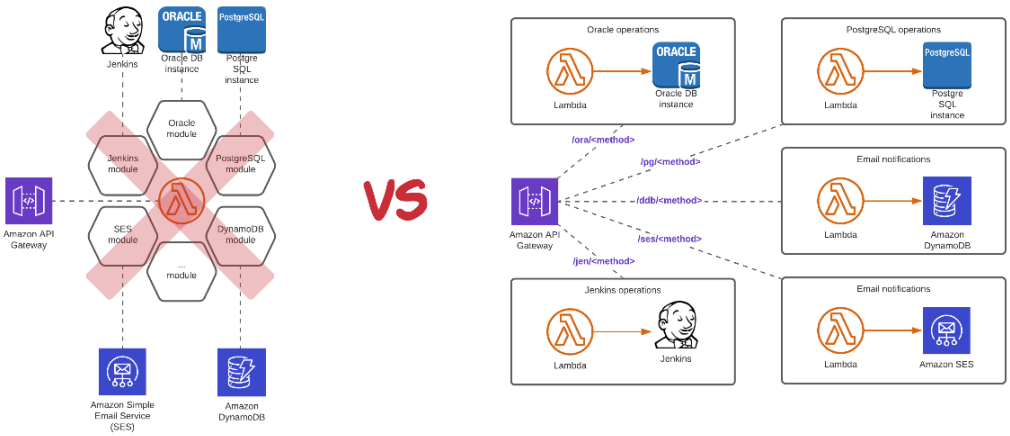

Figure 19: Serverless and Server architecture

Stopping Unauthorised Access:

- Creating a role for DynamoDB such that only it has access to the Lambda function.
- IAM can be used to add an additional level of security.

7. Naming Scheme
   For clarity and organisation, AWS resources such as Auto Scaling Groups (ASGs), Launch Templates, etc must be named consistently. It is easier to identify and manage resources when they are named "application-name-asg-instance.type" for ASGs and "application-name-instance.type" for Launch Templates. This method improves clarity, helps with resource organisation, acts as documentation, and provides scalability as AWS infrastructures increase. All team members must follow the naming standard as it is a good practise in order to maintain an organised and easily readable AWS architecture. (*Https://Peritossolutions.Com/Aws/Aws-Workload-Naming-Convention/*, n.d.)

8. Issues faced overall during the coursework
   When trying autoscaling for different instance types, alarms that have already been set up can be used more than once. There was a time when an error message, "Unable to attach metric alarm to dynamic scaling policy" popped up while setting up dynamic scaling policies. After many failed attempts and more study, it was found out that alarms can only be connected to a maximum of 5 autoscaling groups. If one needs to use the same alarm again, then one of the autoscaling group has to be deleted, or it has to be detached from one of the groups.

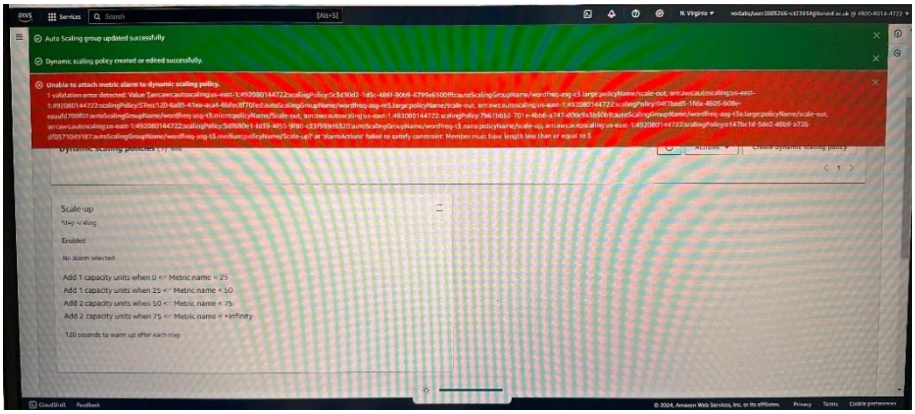

Figure 20: Error: Unable to attach metric alarm to dynamic scaling policy

9. Based on the services taught in the LSDE course, two alternative AWS data processing options that might increase performance and robustness for a word frequency analysis task could include:

- Amazon EC2 with Auto Scaling and Elastic Load Balancing (ELB): AWS EC2 provides scalable computing capability for this solution. EC2 instances let you modify the computing environment to your word frequency application. To handle shifting workloads and maintain performance, use Auto Scaling. This functionality automatically scales EC2 instances up to meet demand and down to optimise costs during calmer periods. ELB would also distribute application traffic over numerous EC2 instances, improving fault tolerance and load balancing. EC2's computational capacity, Auto Scaling's dynamic flexibility, and ELB's traffic management make this system resilient and performant, managing fluctuating processing loads with high availability.

- AWS Lambda with Amazon API Gateway: Serverless design reduces infrastructure administration. AWS Lambda lets you run code in response to data uploads without server upkeep. Running code in parallel and processing triggers independently scales it automatically. With Lambda, the Amazon API Gateway can construct, publish, maintain, monitor, and protect APIs at any scale. This gateway manages all Lambda function API calls, guaranteeing

efficient, safe, and scalable word frequency application access. Serverless computing is scalable and cost-effective since it just pays for compute time. Amazon Lambda and API Gateway provide a highly available, fault-tolerant solution that can withstand abrupt data processing spikes without user intervention, making it suitable for variable and unpredictable workloads.

References:

https://aws.amazon.com/ec2/autoscaling/getting-started/. (n.d.).

https://aws.amazon.com/ec2/instance-types/. (n.d.).

https://aws.amazon.com/ec2/pricing/on-demand/. (n.d.).

https://peritossolutions.com/aws/aws-workload-naming-convention/.(n.d.).